Compiler Construction Supervision 3

Matthew Johnson
Trinity Hall
mjj29@cam.ac.uk

March 11, 2003

1 y1998p3q1

1.1 a.

```
val a = 10;

val f = fn x => x*a;

val a = 1;

val r = f 3;
```

1.2 b.

```
let str = "3";
let num = 4;

let result = str + num;

print result;
```

1.3 c.

`int v[8]` declares a static array of length 8. This reduces to a memory allocation of `8*sizeof(int)`, and a pointer being assigned to `v`. In the expression `int v[e]`, then e must first be evaluated to find out how big an array to allocate, and then the allocation must be done using registers, not constants.

1.4 d.

this is ambiguous: $E \rightarrow (+E)|[-E]|(E)|[E]|E * E|a|b|c$
this isn't: $E \rightarrow F|E * F$
$F \rightarrow (+F)|[-F]|(F)|[F]|a|b|c$

1.5 e.

```
const
   ePlus := 1;
   eMinus := 2;
   eTerm := 3;
   tExp := 1;
   tPrim := 2;
   pBracket := 1;
   pNum := 2;

type
   pExpr : ^Expr;
   Expr : record
      case etype : integer of
         ePlus: begin
            left : pExpr;
            right : pTerm;
         end;
         eMinus: begin
            left : pExpr;
            right : pTerm;
         end;
         eTerm: value : pTerm;
      end;
   end;

   pTerm : ^Term;
   Term : record
      case ttype : integer of
         tExp : begin
            base : pPrim;
            exp : pTerm;
         end;
         tPrim : value : pPrim;
      end;
   end;
```

```
pPrim : ^Prim;
Prim : record
   case ptype : integer of
      pBracket : contents : pExpr;
      pNum : value : integer;
   end;
end;
```

2 y1998p6q5

An ELF object file consists of seceral sections.

The first section is the `.text` section, which contains code. Second is the `.data` section - which contains as static data in the program (such as from `x = 37`). Next are 2 sections which contain relocation information for the code and data sections respectively. These are references to the `.sym` table (see later) and where in the code & data sections they should be relocated to, when the external references are resolved.

The `.sym` table contains all the external references made in the object, and what symbols it can export to other objects, followed by a section with these references in string form.

The example program breaks down as follows:

- `.text` - the definition of the function `int f(int y)`.
- `.data` - the contents of the array `a[10]`, ie `2,3,5,7,11,13,19,23,19"` .
- `.string` - the names `a, b, g` and `f`.
- `.sym` - the links between `.string` and the first 2 sections. IE, the entry fo `a` would point to entry 0 in the `.data` segment. `b` and `g` would be undefined (as external references) and `f` would point to entry 0 in the `.text` segment.

3 y1999p3q4

4 y1999p4q4

This is the function using call by name:

```
int f (int x : name)
{
 ...
 .. x ..
 ...
 .. x ..
}

...
f (e);
...
```

This is the function using only call by value (implemented in pascal with function passing)

```
type
   fn : function () : integer;

function f (x : fn) : integer
begin
   ...
   .. x() ..
   ...
   .. x() ..
   ...
end;

...

function e() : integer
begin
   ....
end;

...

f (e);
```

with using the call by name parameter as both L & R values (implemented using C++ references):

```
typedef (int& f ()) * pfun;

int f (pfun x)
{
   x() = x() +1 ... etc
}

int& e()
{
   return &a
}

....

f ( e );

....
```

call-by-value-result:

```
procedure h( var e1 : sometype )
var
   e2 : sometype := e1;
begin
   ..... (* defn using e2 *)
   .....
   e1 := e2;
end;
```